

CHAPITRE 3 : LE PROBLEME DE L'INTERBLOCAGE

Le but de ce chapitre est de présenter le problème de l'interblocage (deadlock) : les conditions de son apparition, et les différentes méthodes qui ont été proposées pour le traiter.

3.1 PRESENTATION :

Un système informatique possède un nombre fini de ressources qui doivent être distribuées parmi un certain nombre de processus concurrents. Les ressources sont groupées en plusieurs types, lesquels peuvent exister en plusieurs instances identiques. L'espace mémoire, le processeur, les périphériques sont des exemples de types de ressources. Par exemple, si un système a 2 processeurs, on dira que le type de ressource processeur possède 2 instances, et si le système est doté de 5 imprimantes, on dira que le type de ressource imprimante possède 5 instances.

Dans des conditions normales de fonctionnement, un processus ne peut utiliser une ressource qu'en suivant la séquence suivante :

Requête – Utilisation - Libération

- *La requête* : le processus fait une demande pour utiliser la ressource. Si cette demande ne peut pas être satisfaite immédiatement, parce que la ressource n'est pas disponible, le processus demandeur se met en état attente jusqu'à ce que la ressource devienne libre.
- *Utilisation* : Le processus peut exploiter la ressource.
- *Libération* : Le processus libère la ressource qui devient disponible pour les autres processus éventuellement en attente.

Définition de l'interblocage :

Un ensemble de processus est dans une situation d'interblocage si chaque processus de l'ensemble attend un événement qui ne peut être produit que par un autre processus de l'ensemble.

Exemple : Un système possède une instance unique de chacun des deux types de ressources R1 et R2. Un processus P1 détient l'instance de la ressource R1 et un autre processus P2 détient l'instance de la ressource R2. Pour suivre son exécution, P1 a besoin de l'instance de la ressource R2, et inversement P2 a besoin de l'instance de la ressource R1. Une telle situation est une situation d'interblocage.

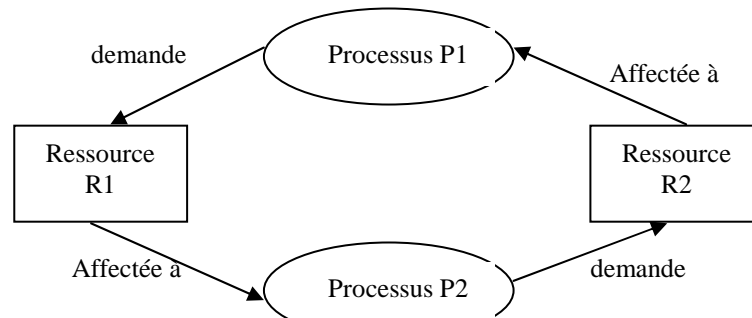


Figure 5.1 Exemple d'interblocage.

Les interblocages sont évidemment indésirables. Dans un interblocage, les processus ne terminent jamais leur exécution et les ressources du système sont immobilisées, empêchant ainsi d'autres travaux de commencer.

Une situation d'interblocage peut survenir si les quatre conditions suivantes se produisent simultanément (Habermann) :

1. *Accès exclusif* : Les ressources ne peuvent être exploitées que par un seul processus à la fois.
2. *Attente et occupation* : Les processus qui demandent de nouvelles ressources gardent celles qu'ils ont déjà acquises et attendent la satisfaction de leur demande
3. *Pas de réquisition* : Les ressources déjà allouées ne peuvent pas être réquisitionnées.
4. *Attente circulaire* : Les processus en attente des ressources déjà allouées forment une chaîne circulaire d'attente.

Définition de la famine :

On dit qu'un processus est dans une situation de famine (starvation) s'il attend indéfiniment une ressource (qui est éventuellement occupée par d'autre processus).

Notons que l'interblocage implique nécessairement une famine, mais le contraire n'est pas toujours vrai.

3.2 GRAPHE D'ALLOCATION DES RESSOURCES :

On peut décrire l'état d'allocation des ressources d'un système en utilisant un graphe. Ce graphe est composé de N nœuds et de A arcs.

L'ensemble des nœuds est partitionné en deux types :

- $P = \{P_1, P_2, \dots, P_m\}$: l'ensemble de tous les processus
- $R = \{R_1, R_2, \dots, R_n\}$ l'ensemble de tous les types de ressources du système

Un arc allant du processus P_i vers un type de ressource R_j est noté $P_i \rightarrow R_j$; il signifie que le processus P_i a demandé une instance du type de ressource R_j . Un arc du type de ressource R_j vers un processus P_i est noté $R_j \rightarrow P_i$; il signifie qu'une instance du type de ressource R_j a été alloué au processus P_i .

Un arc $P_i \rightarrow R_j$ est appelé arc de requête. Un arc $R_j \rightarrow P_i$ est appelé arc d'affectation.

Graphiquement, on représente chaque processus P_i par un cercle et chaque type de ressource R_j comme un rectangle. Puisque chaque type de ressource R_j peut posséder plus d'une instance, on représente chaque instance comme un point dans le rectangle.

Un arc de requête désigne seulement le rectangle R_j , tandis que l'arc d'affectation doit aussi désigner un des points dans le rectangle.

Quand un processus P_i demande une instance du type de ressource R_j , un arc de requête est inséré dans le graphe d'allocation des ressources. Quand cette requête peut être satisfaite, l'arc de requête est instantanément transformé en un arc d'affectation. Quand plus tard, le processus libère la ressource l'arc d'affectation est supprimé.

Exemple : L'état d'allocation d'un système est décrit par les ensembles suivants :

- Ensemble des processus $P = \{P_1, P_2, P_3\}$
- Ensemble des ressources $R = \{R_1, R_2, R_3, R_4\}$
- Ensemble des arcs $A = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

Le nombre d'instances par ressources est donné par ce tableau :

Type de ressources	Nombre d'instances
R1	1
R2	2
R3	2
R4	3

Voici le graphe d'allocation des ressources associé à ce système :

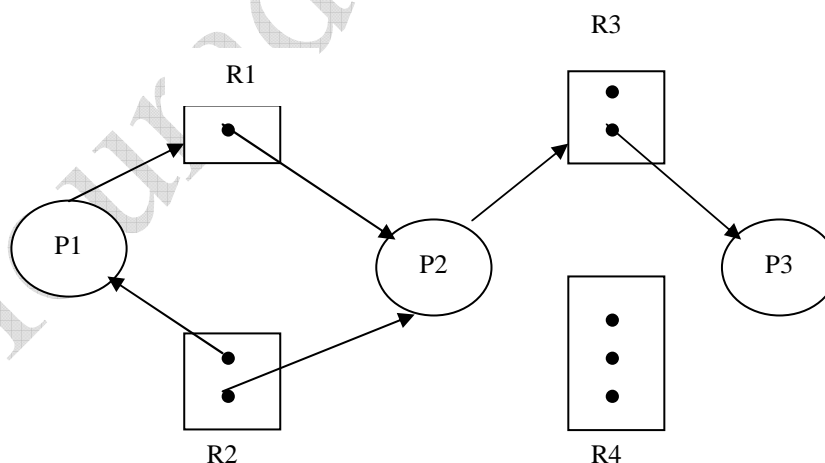


Figure 5.2 Graphe d'allocation des ressources.

On peut faire la lecture suivante sur ce graphe :

- Le processus P1 détient une instance de la ressource R2 et demande une instance de la ressource R1
- Le processus P2 détient une instance de R1 et attend une instance de R3

- Le processus P3 détient une instance de la ressource R3

Si le graphe d'allocation contient un *circuit*, alors il *peut* exister une situation d'interblocage. Considérons les exemples suivants :

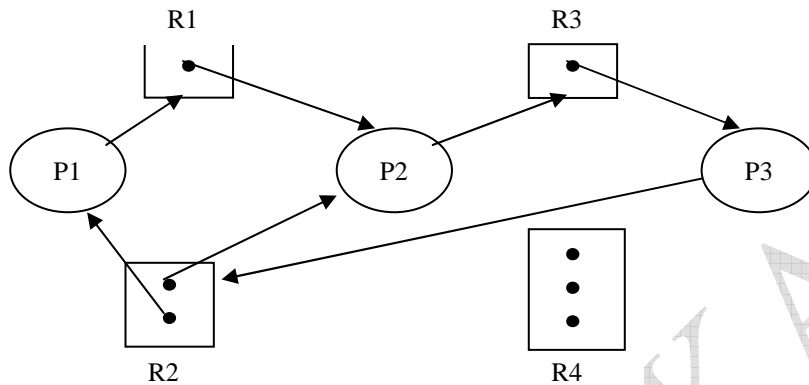


Figure 5.5 Graphe d'allocation des ressources avec circuit et interblocage

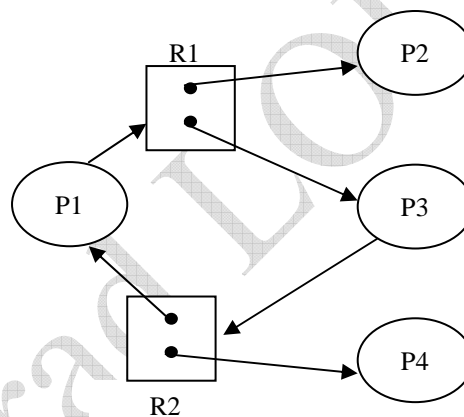


Figure 5.4 Graphe d'allocation des ressources avec circuit sans interblocage

3.3 METHODES DE TRAITEMENT DES INTERBLOCAGES :

Il existe essentiellement trois méthodes pour traiter le problème de l'interblocage : *Prévention*, *Évitement*, *Détection-Guérison*.

- **Prévention** : On élimine complètement le risque d'interblocage en faisant en sorte que l'une des quatre conditions d'apparition ne soit pas vérifiée.
- **Évitement** : On évite le risque d'interblocage en veillant à que le système évolue uniquement entre états sains.
- **Détection-Guérison** : On attend que l'interblocage arrive, on le détecte puis on applique une méthode de guérison.

3.4 PREVENTION DES INTERBLOCAGES :

Les méthodes de prévention des interblocages s'assurent que l'une, au moins, des quatre conditions d'apparition de l'interblocage n'est pas vérifiée.

1. *Exclusion mutuelle* : Un processus ne doit jamais attendre une ressource partageable (exemple : il faut autoriser autant de processus que possible pour la lecture d'un fichier). Cependant, il n'est pas possible de prévenir les interblocages en niant la condition de l'exclusion mutuelle : certaines ressources sont non partageables (exemple : il n'est pas possible de partager un fichier entre plusieurs rédacteurs).
2. *Occupation et attente* : Pour s'assurer que la condition d'occupation et d'attente ne se produit jamais dans le système, on doit garantir qu'à chaque fois qu'un processus qui requiert une ressource, il n'en détient aucune autre. Autrement dit, un processus ne doit demander des ressources supplémentaires qu'après avoir libéré les ressources qu'il occupe déjà.

Par exemple, imaginons un processus qui copie des données d'une unité de bandes vers un fichier du disque, trie le fichier puis imprime le résultat. Si l'on doit demander toutes les ressources au début du processus, celui-ci doit dès le départ requérir l'unité de bande, le fichier du disque et l'imprimante. Ainsi, il gardera l'imprimante pendant toute l'exécution, même s'il n'en a besoin qu'à la fin. L'autre façon de faire, serait d'affecter au processus uniquement l'unité de bande et le fichier sur disque, au début de son exécution. Il copie de l'unité de bande vers le disque et les libère ensuite tous les deux. Le processus doit à nouveau demander le fichier sur disque et l'imprimante. Après avoir copié le fichier sur l'imprimante, il libère ces deux ressources et se termine.

3. *Pas de réquisition* : Pour garantir que cette condition ne soit pas vérifiée, on peut utiliser le protocole suivant : Si un processus détenant certaines ressources en demande une autre qui ne peut pas lui être immédiatement allouée, toutes les ressources actuellement allouées à ce processus sont réquisitionnées. C'est à dire que ses ressources sont implicitement libérées.

Les ressources réquisitionnées sont ajoutées à la liste des ressources pour lesquelles le processus attend. Le processus démarrera seulement quand il pourra regagner ses anciennes ressources, ainsi que les nouvelles qu'il requiert.

Ce protocole présente malheureusement au moins deux inconvénients :

- Lenteur dans l'utilisation des ressources puisqu'on peut allouer plusieurs ressources et ne pas les utiliser pendant longtemps.
 - La famine est possible puisqu'un processus peut être retardé indéfiniment parce que l'une des ressources qu'il demande est occupée.
4. *Attente circulaire* : On peut garantir que la condition de l'attente circulaire ne se vérifie jamais, en imposant un ordre total sur les types de ressources et on forçant chaque processus à demander les ressources dans un ordre croissant d'énumération. Par exemple, on numérote les types de ressources du système, comme suit :
 - O(Unité de bandes)=1
 - O(Unité de disque)=2
 - O(Imprimante)=3
 - O(Lecteur CD-ROM)=4
 - O(Lecteur Disquette)=5
 - ... etc.

Pour prévenir l'interblocage, on peut imaginer le protocole suivant : Un processus ne peut acquérir une ressource d'ordre i que s'il a déjà obtenu toutes les ressources nécessaires d'ordre j ($j < i$). Par exemple, un processus désirent utiliser l'unité de bandes et l'imprimante, doit d'abord demander l'unité de bande ensuite l'imprimante. On peut démontrer que de cette façon, il n'y a aucun risque d'interblocage.

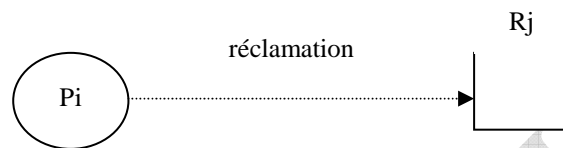
Cependant cet algorithme est inefficace, il peut conduire à une monopolisation inutile d'une ressource (un processus peut détenir une ressource alors qu'il n'en a pas besoin).

3.5 EVITEMENT DES INTERBLOCAGES :

3.5.1 Principe :

Si on dispose d'un système d'allocation de ressources avec une seule instance pour chaque type de ressource, on peut utiliser une variante du graphe d'allocation de ressources pour la prévention des interblocages.

En plus des arcs de requêtes et d'affectation, on introduit un nouveau type d'arc appelé *arc de réclamation*. Un arc de réclamation $P_i \rightarrow R_j$ indique que le processus P_i peut demander la ressource R_j dans le futur (il est représenté en pointillé).

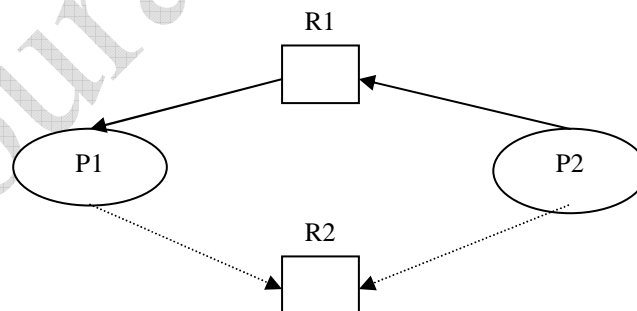


Quand le processus P_i demande réellement la ressource R_j , l'arc de réclamation $P_i \rightarrow R_j$ est transformé en arc de requête. De même, lorsqu'une ressource est libérée par P_i , l'arc d'affectation $R_j \rightarrow P_i$ est reconverti en arc de réclamation $P_i \rightarrow R_j$.

On peut éviter que l'interblocage arrive en empêchant P_i qui demande la ressource R_j , n'obtienne pas cette ressource si la transformation de l'arc de requête $P_i \rightarrow R_j$ en arc d'affectation provoque un circuit dans le graphe.

S'il n'existe pas de circuit, l'allocation de la ressource laissera le système dans un état *sain*. Si l'on trouve un circuit, l'allocation de la ressource laissera le système dans un état *malsain*. Le processus P_i devra donc attendre pour que sa requête soit satisfaite.

Exemple : Soit le graphe d'allocation suivant :



Supposons que P_2 demande R_2 . Bien que R_2 soit libre, on ne peut l'allouer à P_2 , puisque cette action créerait un circuit dans le graphe.

3.5.2 Etat sain :

On vient de voir que le principe des méthodes d'évitement de l'interblocage est de garder le système toujours dans un état sain. Nous donnons maintenant une définition formelle de ce qu'est un état sain :

Définition d'un état sain :

Un système est dans un état sain s'il existe une séquence saine. Une séquence de processus $\langle P_1, P_2, \dots, P_N \rangle$ est une séquence saine pour l'état d'allocation courant si, pour chaque P_i , les requêtes de ressources de P_i peuvent être satisfaites par les ressources couramment disponibles, plus les ressources détenues par tous les P_j , avec $j < i$.

En effet, dans cette situation, si les ressources demandées par P_i ne sont pas immédiatement disponibles, P_i peut attendre jusqu'à la terminaison de tous les P_j . Une fois qu'ils ont fini, P_i peut obtenir les ressources nécessaires, achever sa tâche et rendre les ressources allouées. Quand P_i termine, P_{i+1} peut obtenir les ressources manquantes, et ainsi de suite. Si une telle séquence n'existe pas, on dit que le système est dans un état malsain, c'est à dire qu'il y a un risque d'apparition d'interblocage.

Exemple : un système possède 12 unités de bandes magnétiques et 3 processus P_0 , P_1 et P_2 . On suppose que les besoins des trois processus en ressources sont: 10 pour P_0 , 4 pour P_1 , 9 pour P_2 . D'autre part, on suppose qu'à l'instant t_0 l'état d'allocation des ressources par les processus est le suivant : 5 pour P_0 , 2 pour P_1 et 2 pour P_2 . Le nombre d'unités de bandes libres est donc égal à 3.

Processus	Besoins maximaux	Besoins actuellement satisfaits	Besoins restant à satisfaire
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Nombre de ressources (unités de bandes) disponibles : 03.

A l'instant t_0 , le système est dans un état sain. En effet, la séquence $\langle P_1, P_0, P_2 \rangle$ est saine, puisque l'on peut allouer immédiatement au processus P_1 toutes ses ressources manquantes et les rendre ensuite. Dans ce cas le système disposerait de 5 unités de bandes, le processus P_0 peut donc obtenir toutes ses ressources manquantes et les rendre ensuite. Le système disposerait alors de 10 unités de bandes et enfin le processus P_2 peut obtenir ses unités de bandes et les rendre, le système disposerait donc de 12 unités de bandes.

Il est possible de passer d'un état sain à un état malsain. Supposons qu'à l'instant t_1 le processus P_2 demande et qu'on lui accorde 1 unité de bandes de plus. L'état du système serait alors le suivant :

Processus	Besoins maximaux	Besoins actuellement satisfaits	Besoins restant à satisfaire
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

Nombre de ressources (unités de bandes) disponibles : 02.

Le système n'est plus dans un état sain. En effet, on ne peut allouer toutes ses unités de bandes qu'au processus P_1 . Quand il les rendra, le système ne disposera que de 04 unités de bandes. Comme le processus P_0 peut demander 5 unités et le processus P_2 7 unités, le système n'aura pas suffisamment de ressources pour les satisfaire. Ou pourra alors avoir un cas d'interblocage, puisque P_0 et P_2 seront retardés indéfiniment. Pour éviter cette situation, il ne fallait pas accorder à P_1 la ressource qu'il a demandée ; il fallait le faire attendre.

3.5.3 Algorithme du Banquier :

L'algorithme du banquier (Habermann et Dijkstra) est un algorithme d'évitement des interblocages qui s'applique dans le cas général où chaque type de ressources possède plusieurs instances. Le nom de

l'algorithme a été choisi parce que cet algorithme pourrait s'appliquer dans un système bancaire pour s'assurer que la banque ne prête jamais son argent disponible de telle sorte qu'elle ne puisse plus satisfaire tous ses clients.

Quand un nouveau processus entre dans le système, il doit déclarer le nombre maximal d'instances de chaque type de ressources dont il aura besoin. Ce nombre ne doit pas excéder le nombre total de ressources du système. Au cours de son exécution, quand un processus demande un ensemble de ressources, l'algorithme vérifie si cela gardera toujours le système dans un état sain. Dans l'affirmative la demande est accordée, dans la négative la demande est retardée.

Soient m le nombre de types de ressources du système, et n le nombre de processus. Pour fonctionner, l'algorithme maintient plusieurs structures de données :

- **Available** : C'est un Vecteur de longueur m indiquant le nombre de ressources disponibles de chaque type. Ainsi, si $Available[j]=k$, cela veut dire que le type de ressources R_j possède k instances disponibles.
- **Max** : C'est une matrice $n \times m$ définissant la demande maximale de chaque processus. Ainsi, Si $Max[i, j]=k$, cela veut dire que le processus P_i peut demander au plus k instances du type de ressources R_j .
- **Allocation** : C'est une matrice $n \times m$ définissant le nombre de ressources de chaque type de ressources actuellement alloué à chaque processus. Ainsi si $Allocation[i, j]=k$, cela veut dire que l'on a alloué au processus P_i k instances du type de ressources R_j .
- **Need** : C'est une matrice $n \times m$ indiquant les ressources restant à satisfaire à chaque processus. Ainsi, si $Need[i, j]=k$, cela veut dire que le processus P_i peut avoir besoin de k instances au plus du type de ressources R_j pour achever sa tâche.
- **Request** : C'est une matrice $n \times m$ indiquant les ressources supplémentaires que les processus viennent de demander. Ainsi, si $Request[i, j]=k$, cela veut dire que le processus P_i vient de demander k instances supplémentaires du type de ressources R_j .

De ce qui précède, on peut remarquer que :

1. Ces structures de données peuvent varier dans le temps, en taille et en valeur.
2. La matrice *Need* peut être calculée à partir des matrices *Max* et *Allocation* :

$$Need[i, j] = Max[i, j] - Allocation[i, j]$$

Notations : Pour des raisons pratiques, on utilise les notations suivantes :

- $Allocation_i$: Vecteur désignant les ressources actuellement allouées au processus P_i .
- $Need_i$: Vecteur désignant les ressources supplémentaires que le processus P_i peut encore demander.
- $Request_i$: Vecteur désignant les ressources supplémentaires que le processus P_i vient de demander.
- $X \leq Y$: X et Y sont des vecteurs si $X[i] \leq Y[i]$ pour chaque i allant de 1 à n .

L'algorithme du Banquier peut être scindé en deux parties complémentaires : Un algorithme de requête de ressources et un algorithme de vérification si l'état du système est sain.

Nous donnons maintenant le code de chacune des deux parties.

Algorithme de requête de ressources

Début

/* Cet algorithme est appelé à chaque fois qu'un processus fait une demande de ressources

Etape 1 : Si $Request_i \leq Need_i$
Alors Aller à l'étape 2
Sinon erreur : le processus a excédé ses besoins maximaux
Finsi

Etape 2 : Si $Request_i \leq Available$
Alors Aller à l'étape 3
Sinon Attendre : les ressources ne sont pas disponibles.
Finsi

Etape 3 : Sauvegarder l'état du système (les matrices Available, Allocation et Need).
Allouer les ressources demandées par le processus P_i en modifiant l'état du système de la manière suivante :

$Available := Available - Request_i$
 $Allocation_i := Allocation_i + Request_i$
 $Need_i := Need_i - Request_i$

Si $Verification_Etat_Sain = Vrai$
Alors L'allocation est validée
Sinon L'allocation est annulée ; Restaurer l'ancien Etat du système

Finsi

Fin.

Mourad LOUKKAM